Artículo
Jose Tomas Salvador · Abr 20, 2021 Lectura de 9 min

Conexión con JDBC desde Python a la base de datos de IRIS - una nota rápida

Palabras clave: Python, JDBC, SQL, IRIS, Jupyter Notebook, Pandas, Numpy y aprendizaje automático

Hoy me he encontrado con este artículo de Zphong Li, que publicó en Enero de 2020 pero que creo que es muy interesante y aún útil a día de hoy. Así que... para los que estéis haciendo vuestros primeros pinitos en Machine Learning con InterSystems IRIS, Python y Jupyter... aquí lo tenéis!!

1. Objetivo

Esta es una nota sencilla de 5 minutos, donde os muestro cómo invocar el controlador JDBC de IRIS con la ayuda de Python 3, por ejemplo desde un Jupyter Notebook, para leer y escribir datos en una instancia de la base de datos de IRIS vía SQL.

El año pasado Zhong Li publicó una nota breve sobre como Enlazar Python con una base de datos Caché (sección 4.7). Ahora podría ser el momento de recapitular algunas opciones y discusiones sobre el uso de Python para acceder a una base de datos de IRIS, para leer sus datos en un dataframe de Pandas y una matriz de NumPy para realizar un análisis básico, y después escribir algunos datos pre-procesados o normalizados de nuevo en IRIS y que esté listo para canalizaciones (*pipelines*) adicionales de ML/DL.

Inmediatamente se me ocurren varias **opciones**:

1. **ODBC**: ¿ Cómo hacerlo con PyODBC para Python 3 y SQL nativo?
2. **JDBC**: ¿ Cómo hacerlo con JayDeBeApi para Python 3 y SQL nativo?
3. **Spark**: ¿ Cómo hacerlo con PySpark y SQL?
4. **API nativa de Python para IRIS**: ¿ Qué hay más allá del anterior Python Binding para Caché?
5. **¿ IPtyhon Magic SQL %%sql?** ¿ Podría funcionar con IRIS?

¿ Hay alguna otra opción que se me haya escapado? También estoy interesado en probarla.

2. Alcance

¿ Comenzamos con un enfoque JDBC común? En la siguiente nota breve recapitularemos ODBC, Spark y la API nativa de Python.

En el alcance:

Los siguientes componentes comunes se abordan en esta demostración rápida:

Anaconda
Jupyter Notebook
Python 3
JayDeBeApi

JPyPe
Pandas
NumPy
Una instancia de IRIS 2019.x

Fuera del alcance:

En esta nota rápida NO se abordarán los siguientes aspectos - son importantes y pueden tratarse por separado con soluciones, implementaciones y servicios específicos:

Seguridad de extremo a extremo.
Rendimiento no funcional, etc.
Solución de problemas y soporte.
Licencias.

3. Demostración

3.1 Ejecución de una instancia de IRIS:

Simplemente ejecuté un contenedor IRIS 2019.4 como servidor "remoto" de la base de datos. Puedes utilizar cualquier instancia de IRIS a la que tengas acceso autorizado.

```
zhongli@UKM5530ZHONGLI MINGW64 /c/Program Files/Docker Toolbox
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
NAMES
d86be69a03ab        quickml-demo        "/iris-main"        3 days ago          Up 3 days (healthy)
0.0.0.0:9091->51773/tcp, 0.0.0.0:9092->52773/tcp   quickml
```

3.2 Anaconda y Jupyter Notebook:

Reutilizaremos el mismo enfoque de configuración descrito aquí para Anaconda (sección 4.1) y aquí para Jupyter Notebook (sección 4) en un ordenador portátil. Python 3.x se instala junto con este paso.

3.3 Instalar JayDeBeApi y JPyPe:

Inicié mi JupyterNotebook, y luego simplemente ejecuté lo siguiente en sus celdas para **configurar un puente desde Python-a-JDBC/Java**:

```
!conda install --yes -c conda-forge jaydebeapi
```

JayDeBeApi utiliza JPype 0.7 en el momento de escribir este artículo (enero del 2020), pero no funciona debido a un error conocido, por lo que tuve que utilizar la 0.6.3

```
!conda install --yes -c conda-forge JPype1=0.6.3 --force-reinstall
```

3.4 Conectar a una base de datos de IRIS por medio de JDBC

Hay una documentación oficial de JDBC en IRIS aquí.

Para las ejecuciones de Python SQL sobre JDBC, utilicé los siguientes códigos como ejemplo. Se conecta a una tabla de datos llamada "DataMining.IrisDataset" dentro del namespace "USER" de esta instancia de IRIS.

```
### 1. Establezca las variables de entorno, si es necesario
#importa os
```

```
#os.environ['JAVA_HOME']='C:\Progra~1\Java\jdk1.8.0_241'
#os.environ['CLASSPATH'] = 'C:\interSystems\IRIS20194\dev\java\lib\JDK18\intersystems-
jdbc-3.0.0.jar'
#os.environ['HADOOP_HOME']='C:\hadoop\bin'
#winutil binary must be in Hadoop's Home



### 2. Obtiene la conexión JDBC y el cursor
import JayDeBeApi
url = "jdbc:IRIS://192.168.99.101:9091/USER"
driver = 'com.intersystems.jdbc.IRISDriver'
user = "SUPERUSER"
password = "SYS"
#libx = "C:/InterSystems/IRIS20194/dev/java/lib/JDK18"
jarfile = "C:/InterSystems/IRIS20194/dev/java/lib/JDK18/intersystems-jdbc-3.0.0.jar"
conn = jaydebeapi.connect(driver, url, [user, password], jarfile)
curs = conn.cursor()



### 3. Especifica la fuente de la tabla de datos
dataTable = "DataMining.IrisDataset"



### 4. Obtiene el resultado y visualiza
curs.execute("select TOP 20 * from %s" % dataTable)
result = curs.fetchall()
print("Total records: " + str(len(result)))
for i in range(len(result)):
    print(result[i])



### 5. Cerrar y limpiar - Los mantendre abiertos para los próximos accesos.
#curs.close()
#conn.close()
```

```
Total records: 150
(1, 1.4, 0.2, 5.1, 3.5, 'Iris-setosa')
(2, 1.4, 0.2, 4.9, 3.0, 'Iris-setosa')
(3, 1.3, 0.2, 4.7, 3.2, 'Iris-setosa')
... ...
(49, 1.5, 0.2, 5.3, 3.7, 'Iris-setosa')
(50, 1.4, 0.2, 5.0, 3.3, 'Iris-setosa')
(51, 4.7, 1.4, 7.0, 3.2, 'Iris-versicolor')
... ...
(145, 5.7, 2.5, 6.7, 3.3, 'Iris-virginica')
... ...
(148, 5.2, 2.0, 6.5, 3.0, 'Iris-virginica')
(149, 5.4, 2.3, 6.2, 3.4, 'Iris-virginica')
(150, 5.1, 1.8, 5.9, 3.0, 'Iris-virginica')
```

Ahora hemos verificado que Python en JDBC estaba funcionando. Lo siguiente es solo un poco de análisis de

datos de rutina y preprocesamiento para los canales habituales de ML que deberíamos mencionar una y otra vez para demostraciones y comparaciones posteriores, por lo que se adjunta para mayor comodidad.

3.5 Convertir los resultados de SQL al DataFrame de Pandas y después a las matrices de NumPy

Instala los paquetes Pandas y NumPy a través de Conda si aún no están instalados, como se explicó en el punto 3.3.

A continuación, ejecute lo siguiente como un ejemplo:

```
### Transforma los resultados de SQL "sqlData" en un dataframe de Pandas "df", y desp
ués en una matriz de NumPy "arrayN" para otras canalizaciones (*pipelines*) de ML
import pandas as pd
sqlData = "SELECT * from DataMining.IrisDataset"
df= pd.io.sql.read_sql(sqlData, conn)
df = df.drop('ID', 1)
df = df[['SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth', 'Species']]



# Establece las etiquetas en 0, 1, 2, para la matriz NumPy
df.replace('Iris-setosa', 0, inplace=True)
df.replace('Iris-versicolor', 1, inplace=True)
df.replace('Iris-virginica', 2, inplace=True)



# Convierte el dataframe en una matriz Numpy
arrayN = df.to_numpy()



### 6. Cierra y limpia - ¿si la conexión ya no es necesaria?
#curs.close()
#conn.close()
```

Echemos un vistazo rutinario a los datos actuales:

```
df.head(5)
```

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
df.describe()
```

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667	1.000000
std	0.828066	0.433594	1.764420	0.763161	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

Ahora tenemos un DataFrame, y una matriz NumPy normalizada de una tabla de datos fuente a nuestra disposición.

Ciertamente, ¿ podemos probar varios análisis de rutina con los que comenzaría un experto en ML, como se indica a continuación, en Python para reemplazar a R, como en el enlace que se encuentra aquí?

La fuente de datos se cita aquí

3.6 Dividir los datos y volver a escribirlos en la base de datos de IRIS por medio de SQL:

Ciertamente, podemos dividir los datos en un conjunto de Entrenamiento y otro de Validación o Prueba, como es habitual, y después escribirlos de nuevo en tablas temporales de la base de datos, para algunas emocionantes funciones ML de IRIS:

```
import numpy as np
from matplotlib
import pyplot
from sklearn.model_selection import train_test_split
```

```
# mantiene, por ejemplo, el 20% = 30 filas como datos de prueba, y el 80% = 120 filas
 para entrenamiento
X = arrayN[:,0:4]
y = arrayN[:,4]
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y, test_size=0.20,
 random_state=1, shuffle=True)



# Hace que el 80% de las filas, escogidas aleatoriamente, estén en el conjunto Entren
amiento
labels1 = np.reshape(Y_train,(120,1))
train = np.concatenate([X_train, labels1],axis=-1)



# Hace que el 20% de las filas, escogidas eleatoriamente, estén en el conjunto Prueba
,
lTest1 = np.reshape(Y_validation,(30,1))
test = np.concatenate([X_validation, lTest1],axis=-1)



# Escribe el conjunto de datos de entrenamiento en un dataframe de Pandas
dfTrain = pd.DataFrame({'SepalLength':train[:, 0], 'SepalWidth':train[:, 1], 'PetalLe
ngth':train[:, 2], 'PetalWidth':train[:, 3], 'Species':train[:, 4]})
dfTrain['Species'].replace(0, 'Iris-setosa', inplace=True)
dfTrain['Species'].replace(1, 'Iris-versicolor', inplace=True)
dfTrain['Species'].replace(2, 'Iris-virginica', inplace=True)



# Escribe los datos de pruebas en un dataframe de Pandas
dfTest = pd.DataFrame({'SepalLength':test[:, 0], 'SepalWidth':test[:, 1], 'PetalLengt
h':test[:, 2], 'PetalWidth':test[:, 3], 'Species':test[:, 4]})
dfTest['Species'].replace(0, 'Iris-setosa', inplace=True)
dfTest['Species'].replace(1, 'Iris-versicolor', inplace=True)
dfTest['Species'].replace(2, 'Iris-virginica', inplace=True)



### 3. Especifica los nombres de las tablas temporales
dtTrain = "TRAIN02"
dtTest = "TEST02"



### 4. Crea 2 tablas temporales - puedes probar eliminar las tablas temporales y volv
er a crearlas una y otra vez
curs.execute("Create Table %s (%s DOUBLE, %s DOUBLE, %s DOUBLE, %s DOUBLE, %s VARCHAR
(100))" %(dtTrain,dfTrain.columns[0],dfTrain.columns[1],dfTrain.columns[2], dfTrain.c
olumns[3], dfTrain.columns[4]))
curs.execute("Create Table %s (%s DOUBLE, %s DOUBLE, %s DOUBLE, %s DOUBLE, %s VARCHAR
(100))" %(dtTest,dfTest.columns[0],dfTest.columns[1],dfTest.columns[2],dfTest.columns
[3],dfTest.columns[4]))
```

```
### 5. Escribe el conjunto de Entrenamiento y el conjunto de Prueba en las tablas. Se
 puede intentar borrar cada vez el registro anterior y luego insertarlo otra vez.
curs.fast_executemany = True
curs.executemany( "INSERT INTO %s (SepalLength, SepalWidth, PetalLength, PetalWidth,
Species) VALUES (?, ?, ?, ? ,?)" % dtTrain, list(dfTrain.itertuples(index=False, name
=None)) )
curs.executemany( "INSERT INTO %s (SepalLength, SepalWidth, PetalLength, PetalWidth,
Species) VALUES (?, ?, ?, ? ,?)" % dtTest, list(dfTest.itertuples(index=False, name=N
one)) )



### 6. Cierra y limpia - ¿si la conexión ya no es necesaria?
#curs.close()
#conn.close()
```

Ahora, si cambiamos a la Consola de Administración de IRIS, o a la Consola del Terminal SQL, deberíamos ver 2 tablas temporales creadas: TRAIN02 con 120 filas y TEST02 con 30 filas.

Tendré que detenerme aquí, ya que supuestamente este artículo es una nota rápida muy breve.

4. Advertencias

- El contenido anterior puede ser modificado o perfeccionado.

5. Siguiente

Simplemente reemplazaremos las secciones 3.3 y 3.4 con PyODBC, PySPark y la API nativa de Python para IRIS, a menos que a alguien no le importe contribuir con una nota rápida - también lo agradeceré.

#JDBC #Machine learning #ODBC #Python #InterSystems IRIS

URL de fuente: https://es.community.intersystems.com/post/conexi%C3%B3n-con-jdbc-desde-python-la-base-de-datos-de-iris-una-nota-r%C3%A1pida